UDA
Laboratorio I

Unidad I
Introducción

Paula C. Martinez

Objetivos de la asignatura

Aprender a manejar con habilidad lenguajes orientados a eventos para el desarrollo de aplicaciones y programar sistemas con acceso a datos.

- Los *lenguajes orientados a eventos* es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.
- El creador de un programa dirigido por eventos debe definir los eventos que manejarán su programa y las acciones que se realizarán al producirse cada uno de ellos, lo que se conoce como el administrador de evento.
- Los eventos soportados estarán determinados por el *lenguaje de programación* utilizado, por el sistema operativo e incluso por eventos creados por el mismo programador.

¿Qué veremos?

- Sistema operativo básico. Archivo. Directorios. Comandos internos y externos. Configuración. Tipos de datos. Operadores y expresiones. Asignación. Lectura y escritura. Estructura de control. Repetitivas. Punteros.

- Funciones. Arreglos. Estructuras. Funciones de librería. Archivos de cabeceras. Con desarrollo en un lenguaje propuesto por la cátedra.

- Balance entre tiempo y espacio en los algoritmos. Análisis de complejidad de algoritmos. Verificación de algoritmos y uso de heurísticas en algoritmos.

Introducción

Una *computadora* es un dispositivo electrónico utilizado para procesar información y obtener resultados. Los datos y la información se pueden introducir en la computadora por la entrada (input) y a continuación se procesan para producir una salida (output, resultados).

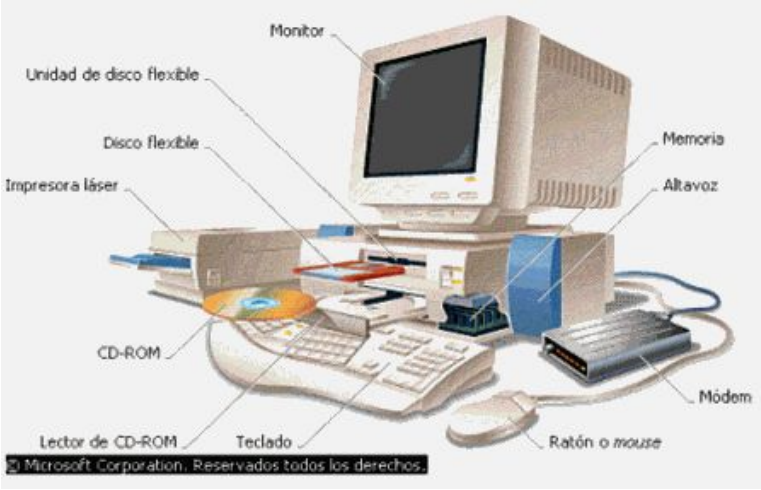
La computadora se puede considerar como una unidad en la que se ponen ciertos datos, entrada de datos, procesa estos datos y produce unos datos de salida.

Introducción

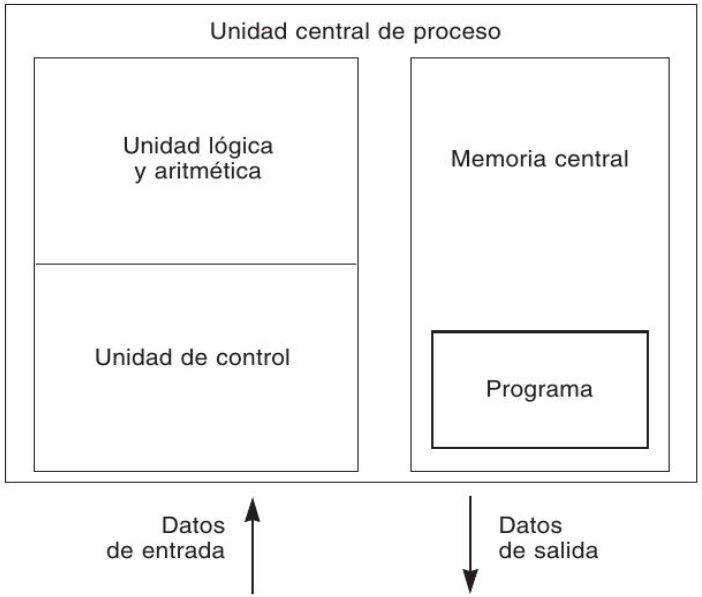
- Los datos de entrada y los datos de salida pueden ser realmente cualquier cosa, texto, dibujos o sonido.
- El sistema más sencillo de comunicarse una persona con la computadora es esencialmente mediante un ratón (mouse), un teclado y una pantalla (monitor).
- Hoy día existen otros dispositivos muy populares tales como escáneres, micrófonos, altavoces, cámaras de vídeo, cámaras digitales, etc.; de igual manera, mediante módems, es posible conectar su computadora con otras computadoras a través de redes, siendo la más importante, la red Internet.

Introducción

- Los componentes físicos que constituyen la computadora, junto con los dispositivos que realizan las tareas de entrada y salida, se conocen con el término *hardware*.
- El conjunto de instrucciones que hacen funcionar a la computadora se denomina programa, que se encuentra almacenado en su memoria; a la persona que escribe programas se llama *programador* y al conjunto de programas escritos para una computadora se llama *software*.

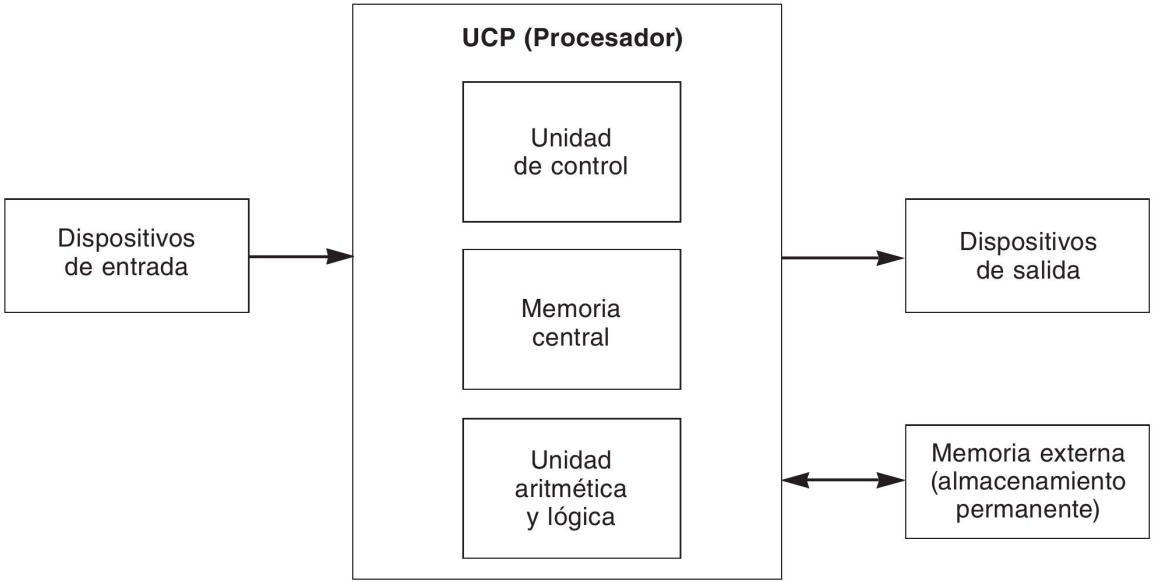


Organización física de la computadora (hardware)



Las computadoras constan fundamentalmente de tres componentes principales: *Unidad Central de Proceso (UCP) o procesador (compuesta de la UAL, Unidad Aritmética y Lógica, y la UC, Unidad de Control); la memoria principal o central y el programa*

Organización física de la computadora (hardware)



Si a la organización física de la figura anterior se le añaden los dispositivos para comunicación con la computadora, aparece la estructura típica de un sistema de computadora: dispositivos de entrada, dispositivos de salida, memoria externa y el procesador/memoria central con su programa.

La memoria central (interna)

- La memoria central o simplemente memoria (interna o principal) se utiliza para almacenar información (RAM, Random, Access Memory).

- En general, la información almacenada en memoria puede ser de dos tipos: instrucciones, de un programa y datos con los que operan las instrucciones.

- Por ejemplo, para que un programa se pueda ejecutar (correr, rodar, funcionar..., en inglés, run), debe ser situado en la memoria central, en una operación denominada carga (load) del programa.

La memoria central (interna)

Después, cuando se ejecuta (se realiza, funciona) el programa, cualquier dato a procesar por el programa se debe llevar a la memoria mediante las instrucciones del programa.

En la memoria central, hay también datos diversos y espacio de almacenamiento temporal que necesita el programa cuando se ejecuta a fin de poder funcionar.

La memoria central (interna)

Con el objetivo de que el procesador pueda obtener los datos de la memoria central más rápidamente, normalmente todos los procesadores actuales (muy rápidos) utilizan una memoria denominada *caché* que sirve para almacenamiento intermedio de datos entre el procesador y la memoria principal.

La memoria caché —en la actualidad— se incorpora casi siempre al procesador.

Organización de la memoria

- La memoria central de una computadora es una zona de almacenamiento organizada en centenares o millares de unidades de almacenamiento individual o celdas.
- La memoria central consta de un conjunto de celdas de memoria (estas celdas o posiciones de memoria se denominan también palabras, aunque no “guardan” analogía con las palabras del lenguaje).
- El número de celdas de memoria de la memoria central, dependiendo del tipo y modelo de computadora; hoy día el número suele ser millones (512, 1.024, etc.). Cada celda de memoria consta de un cierto número de bits (normalmente 8, un byte).

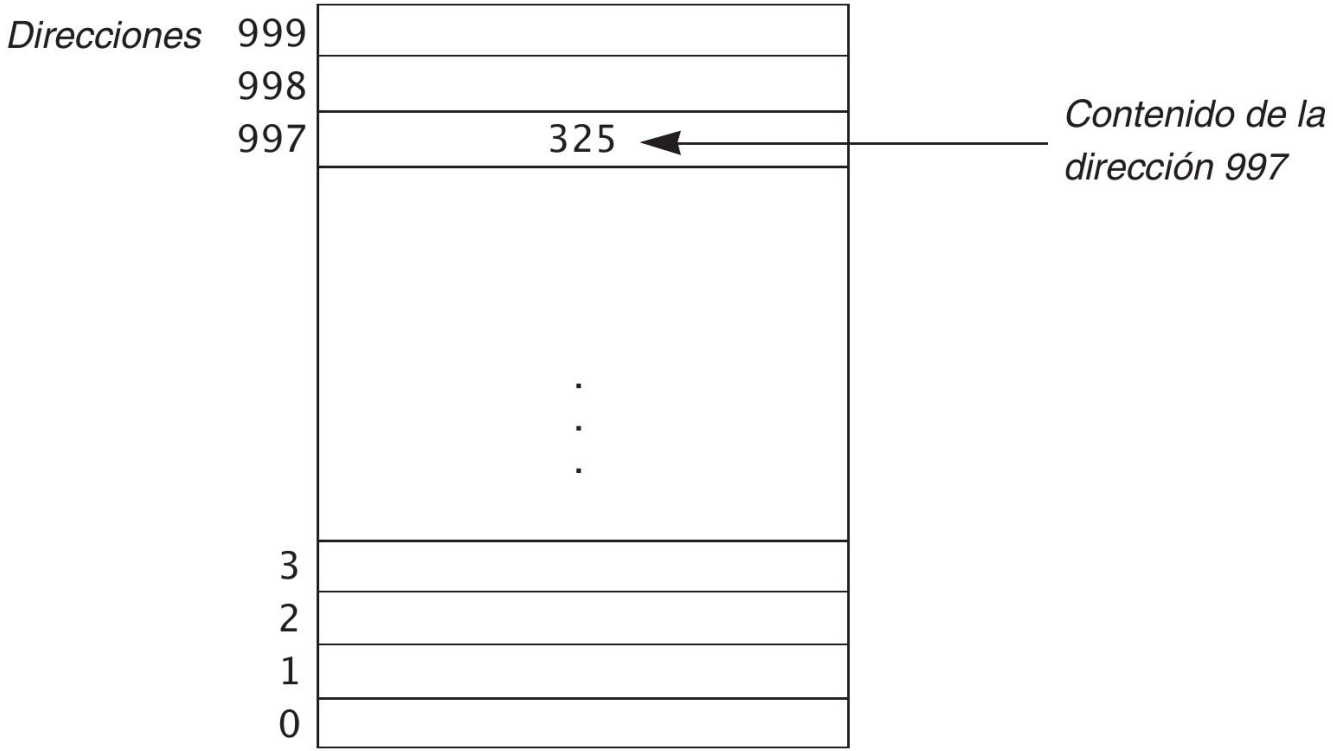
Organización de la memoria

- La unidad elemental de memoria se llama byte (octeto). Un byte tiene la capacidad de almacenar un carácter de información, y está formado por un conjunto de unidades más pequeñas de almacenamiento denominadas bits, que son dígitos binarios (0 o 1). Por definición, se acepta que un byte contiene ocho bits.
- Si queremos almacenar el número del pasaporte P57487891, se ocuparán 9 bytes, pero si se almacena como P5-748-7891, ocupará 11 bytes.
- Estos datos se llaman alfanuméricos, y pueden constar de letras del alfabeto, dígitos o incluso caracteres especiales (símbolos: $, #, *, etc.).
- Mientras que cada carácter de un dato alfanumérico se almacena en un byte, la información numérica se almacena de un modo diferente. Los datos numéricos ocupan 2, 4 e incluso 8 bytes consecutivos, dependiendo del tipo de dato numérico.

Organización de la memoria

- Existen dos conceptos importantes asociados a cada celda o posición de memoria: *su dirección y su contenido*.
- Cada celda o byte tiene asociada una única dirección que indica su posición relativa en memoria y mediante la cual se puede acceder a la posición para almacenar o recuperar información.
- La información almacenada en una posición de memoria es su contenido.
- La figura siguiente muestra una memoria de computadora que consta de 1.000 posiciones en memoria con direcciones de 0 a 999. El contenido de estas direcciones o posiciones de memoria se llaman palabras, de modo que existen palabras de 8, 16, 32 y 64 bits. Por consiguiente, si trabaja con una máquina de 32 bits, significa que en cada posición de memoria de su computadora puede alojar 32 bits, es decir, 32 dígitos binarios, bien ceros o unos.

Organización de la memoria



Organización de la memoria

Siempre que se almacena una nueva información en una posición, se destruye (desaparece) cualquier información que en ella hubiera y no se puede recuperar. La dirección es permanente y única, el contenido puede cambiar mientras se ejecuta un programa.

La memoria central de una computadora puede tener desde unos centenares de millares de bytes hasta millones de bytes. Como el byte es una unidad elemental de almacenamiento, se utilizan múltiplos de potencia de 2 para definir el tamaño de la memoria central: Kilo-byte (KB) igual a 1.024 bytes (2 a la 10) —prácticamente se consideran 1.000—; Megabyte (MB) igual a 1.024 × 1.024 bytes = = 1.048.576 (2 a la 20) —prácticamente se consideran 1.000.000; Gigabyte (GB) igual a 1.024 MB (2 a la 30), 1.073.741.824 = prácticamente se consideran 1.000 millones de MB.

Organización de la memoria

En la memoria principal se almacenan:

- Los datos enviados para procesarse desde los dispositivos de entrada.
- Los programas que realizarán los procesos.
- Los resultados obtenidos preparados para enviarse a un dispositivo de salida.

Representación de la información en la computadora

- Una computadora es un sistema para procesar información de modo automático. Un tema vital en el proceso de funcionamiento de una computadora es estudiar la forma de representación de la información en dicha computadora.
- Es necesario considerar cómo se puede codificar la información en patrones de bits que sean fácilmente almacenables y procesables por los elementos internos de la computadora.
- Las formas de información más significativas son: textos, sonidos, imágenes y valores numéricos y,cada una de ellas presentan peculiaridades distintas. Otros temas importantes en el campo de la programación se refieren a los métodos de detección de errores que se puedan producir en la transmisión o almacenamiento de la información y a las técnicas y mecanismos de comprensión de información al objeto de que ésta ocupe el menor espacio en los dispositivos de almacenamiento y sea más rápida su transmisión.

Representación de textos

La información en formato de texto se representa mediante un código en el que cada uno de los distintos símbolos del texto (tales como letras del alfabeto o signos de puntuación) se asignan a un único patrón de bits. El texto se representa como una cadena larga de bits en la cual los sucesivos patrones representan los sucesivos símbolos del texto original.

Se puede representar cualquier información escrita (texto) mediante caracteres. Los caracteres que se utilizan en computación suelen agruparse en cinco categorías:

Representación de textos

1. Caracteres alfabéticos (letras mayúsculas y minúsculas, en una primera versión del abecedario inglés).

    A, B, C, D, E, ... X, Y, Z, a, b, c, ... , X, Y, Z

2. Caracteres numéricos (dígitos del sistema de numeración).

    0, 1, 2, 3, 4, 5, 6, 7, 8, 9 sistema decimal

3. Caracteres especiales (símbolos ortográficos y matemáticos no incluidos en los grupos anteriores).

    { } Ñ ñ ! ? & > # ç ...

4. Caracteres geométricos y gráficos (símbolos o módulos con los cuales se pueden representar cuadros, figuras geométricas, iconos, etc.

    | —| ||—— ♠ ...

5. Caracteres de control (representan órdenes de control como el carácter para pasar a la siguiente línea [NL] o para ir al comienzo de una línea [RC, retorno de carro, «carriage return, CR»] emitir un pitido en el terminal [BEL], etc.).

Representación de textos

Al introducir un texto en una computadora, a través de un periférico, los caracteres se codifican según un código de entrada/salida de modo que a cada carácter se le asocia una determinada combinación de n bits.

Los códigos más utilizados en la actualidad son: EBCDIC, ASCII y Unicode.

Representación de valores numéricos

- Al introducir un número en la computadora se codifica y se almacena como un texto o cadena de caracteres, pero dentro del programa a cada dato se le envía un tipo de dato específico y es tarea del programador asociar cada dato al tipo adecuado correspondiente a las tareas y operaciones que se vayan a realizar con dicho dato.
- El método práctico realizado por la computadora es que una vez definidos los datos numéricos de un programa, una rutina (función interna) de la biblioteca del compilador (traductor) del lenguaje de programación se encarga de transformar la cadena de caracteres que representa el número en su notación binaria.
- Existen dos formas de representar los datos numéricos: números enteros o números reales.

Representación de enteros

Los datos de tipo entero se representan en el interior de la computadora en notación binaria. La memoria ocupada por los tipos enteros depende del sistema, pero normalmente son dos, bytes (en las versiones de MS-DOS y versiones antiguas de Windows y cuatro bytes en los sistemas de 32 bits como Windows o Linux). Por ejemplo, un entero almacenado en 2 bytes (16 bits):

1000 1110 0101 1011

Los enteros se pueden representar con signo (signed, en C++) o sin signo (unsigned, en C++); es decir, números positivos o negativos. Normalmente, se utiliza un bit para el signo.

Representación de reales

Los números reales son aquellos que contienen una parte decimal como 2,6 y 3,14152. Los reales se representan en notación científica o en coma flotante; por esta razón en los lenguajes de programación, como C++, se conocen como números en coma flotante.

Existen dos formas de representar los números reales. La primera se utiliza con la notación del punto decimal (ojo en el formato de representación español de números decimales, la parte decimal se representa por coma).

Ejemplos: 12.35, 99901.32, 0.00025, 9.0

La segunda forma para representar números en coma flotante en la notación científica o exponencial, conocida también como notación E. Esta notación es muy útil para representar números muy grandes o muy pequeños.

Representación de caracteres

Un documento de texto se escribe utilizando un conjunto de caracteres adecuado al tipo de documento.

En los lenguajes de programación se utilizan, principalmente, dos códigos de caracteres. El más común es ASCII (American Standard Code for Information Interchange) y algunos lenguajes, tal como Java, utilizan Unicode (www.unicode.org). Ambos códigos se basan en la asignación de un código numérico a cada uno de los tipos de caracteres del código.

Tipos de datos enteros y reales

		Carácter y bool
Tipo	**Tamaño**	**Rango**
`short (short int)`	2 bytes	`–32.738..32.767`
`int`	4 bytes	`–2.147.483.648` a `2.147.483.647`
`long (long int)`	4 bytes	`–2.147.483.648` a `2.147.483.647`
`float (real)`	4 bytes	10^{-38} a 10^{38} (*aproximadamente*)
`double`	8 bytes	10^{-308} a 10^{308} (*aproximadamente*)
`long double`	10 bytes	10^{-4932} a 10^{4932} (*aproximadamente*)
`char (carácter)`	1 byte	Todos los caracteres ASCII
`bool`	1 byte	`True` (verdadero) y `false` (falso)

Programas secuenciales, interactivos y orientados a eventos

- Los programas secuenciales o procedimentales son aquellos que inician, reciben los datos de entrada que necesita para ejecutarse, realiza el procesamiento y cálculos, y finalmente guarda los resultados obtenidos.
- No necesita intervención del usuario mientras se está ejecutando
- Los programas interactivos necesitan de la intervención del usuario mientras se están ejecutando, para ingresar datos o para indicar qué acciones deben realizar.

Programas secuenciales, interactivos y orientados a eventos

- Los programas orientados a eventos son tal vez a los que más estamos acostumbrados (Word, Excel, Chrome, Firefox).
- Estos programas una vez que están corriendo, necesitan que el usuario realice acciones para su funcionamiento, estas acciones se las denomina “eventos”.
- Estos programas pasan la mayor parte de su tiempo esperando las acciones del usuario (eventos) y respondiendo a ellas.
- Las acciones que el usuario puede realizar en un momento determinado son diversas, y requieren de un tipo especial de programación: la programación orientada a eventos.
- Lenguajes como Visual Basic y Visual Studio .NET utilizan este tipo de programación.

Lenguajes imperativos o procedimentales

- El paradigma imperativo o procedimental representa el enfoque o método tradicional de programación.
- Un lenguaje imperativo es un conjunto de instrucciones que se ejecutan una por una, de principio a fin, de modo secuencial excepto cuando intervienen instrucciones de salto de secuencia o control.
- Este paradigma define el proceso de programación como el desarrollo de una secuencia de órdenes (comandos) que manipulan los datos para producir los resultados deseados.
- Por consiguiente, el paradigma imperativo señala un enfoque del proceso de programación mediante la realización de un algoritmo que resuelve de modo manual el problema y a continuación expresa ese algoritmo como una secuencia de órdenes.
- En un lenguaje procedimental cada instrucción es una orden u órdenes para que la computadora realice alguna tarea específica.
- Ejemplos: C, Fortran, Pascal, BASIC.

Lenguajes declarativos

- En contraste con el paradigma imperativo el paradigma declarativo solicita al programador que describa el problema en lugar de encontrar una solución algorítmica al problema;
- Es decir, un lenguaje declarativo utiliza el principio del razonamiento lógico para responder a las preguntas o cuestiones consultadas.
- Se basa en la lógica formal y en el cálculo de predicados de primer orden.
- El razonamiento lógico se basa en la deducción.
- Ejemplos: Prolog

Lenguajes funcionales

- Este es un paradigma de programación declarativa basado en el uso de funciones matemáticas,
- La programación funcional tiene sus raíces en el cálculo lambda, un sistema formal desarrollado en los años 1930 para investigar la definición de función, la aplicación de las funciones y la recursión.
- Ejemplos: Clojure, Erlang, Haskell.

Lenguajes orientados a objetos

- Enfoque totalmente distinto al proceso procedimental.
- El enfoque orientado a objetos guarda analogía con la vida real.
- El desarrollo de software OO se basa en el diseño y construcción de objetos que se componen a su vez de datos y operaciones que manipulan esos datos.
- El programador define en primer lugar los objetos del problema y a continuación los datos y operaciones que actuarán sobre esos datos.
- Las ventajas de la programación orientada a objetos se derivan esencialmente de la estructura modular existente en la vida real y el modo de respuesta de estos módulos u objetos a mensajes o eventos que se producen en cualquier instante.
- Ejemplos: C++, Java, Python, Ruby, Smalltalk

Modo de Diseño y Modo de Ejecución

- Mientras uno escribe código, puede diseñar como se va a comportar la aplicación. A esta etapa se la denomina “modo de diseño”.
- Este modo generalmente se lo asociaba con la programación de interfaces gráficas con lenguajes como Visual Basic.

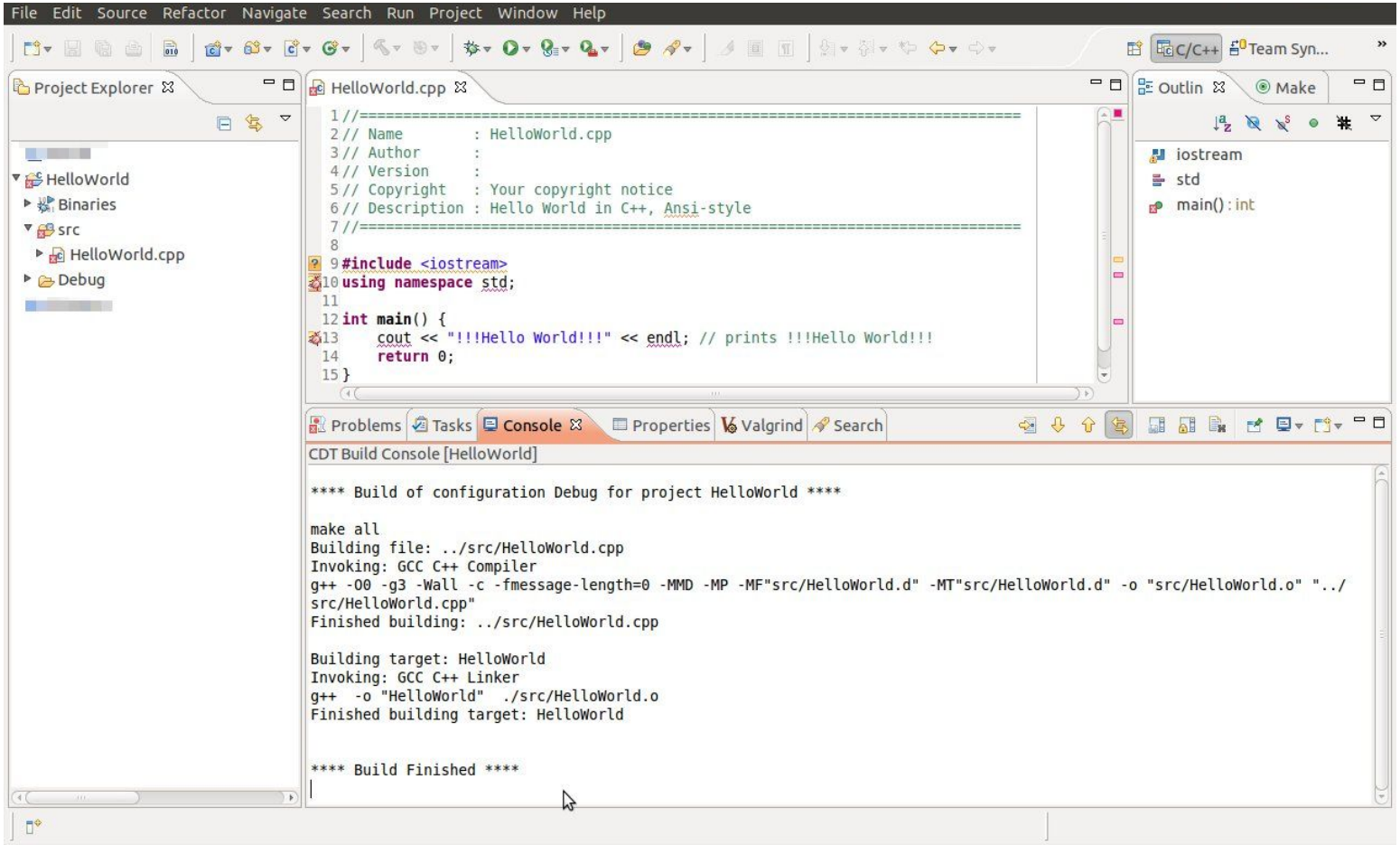
- El “modo de ejecución” es el momento en el que la aplicación está siendo ejecutado por el usuario o el propio programador.
- En este modo la única forma de modificar el comportamiento de la aplicación es a través de su propio código (previamente escrito por el programador).

Entorno Integrado de Desarrollo

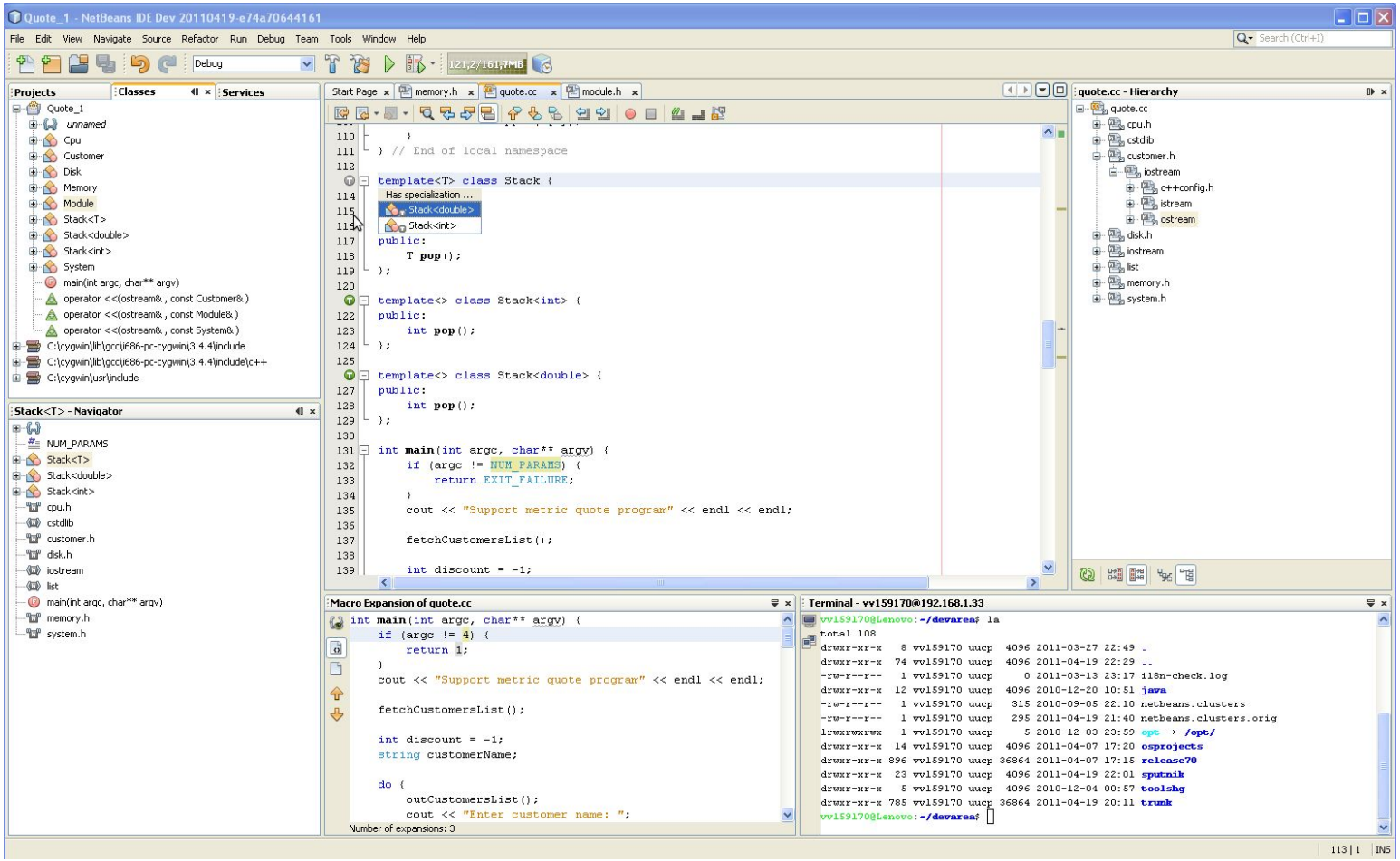
- Un Entorno de Desarrollo Integrado (IDE o Integrated Development Environment) es una aplicación visual que sirve para la construcción de aplicaciones a partir de componentes.
- En general, todas las IDEs contienen:
  - Menú con botones para activar/desactivar componentes de la IDE.
  - Un marco o contenedor para colocar los componentes de la IDE.
  - Buscador de componentes
  - Editor de textos para escribir codigo
  - Panel para navegar por los archivos/directorios del software que se está desarrollando
  - Acceso a interpretes, compiladores y depuradores.
  - Acceso a herramientas de control de versiones (GIT o SVN por ejemplo).

Entorno Integrado de Desarrollo

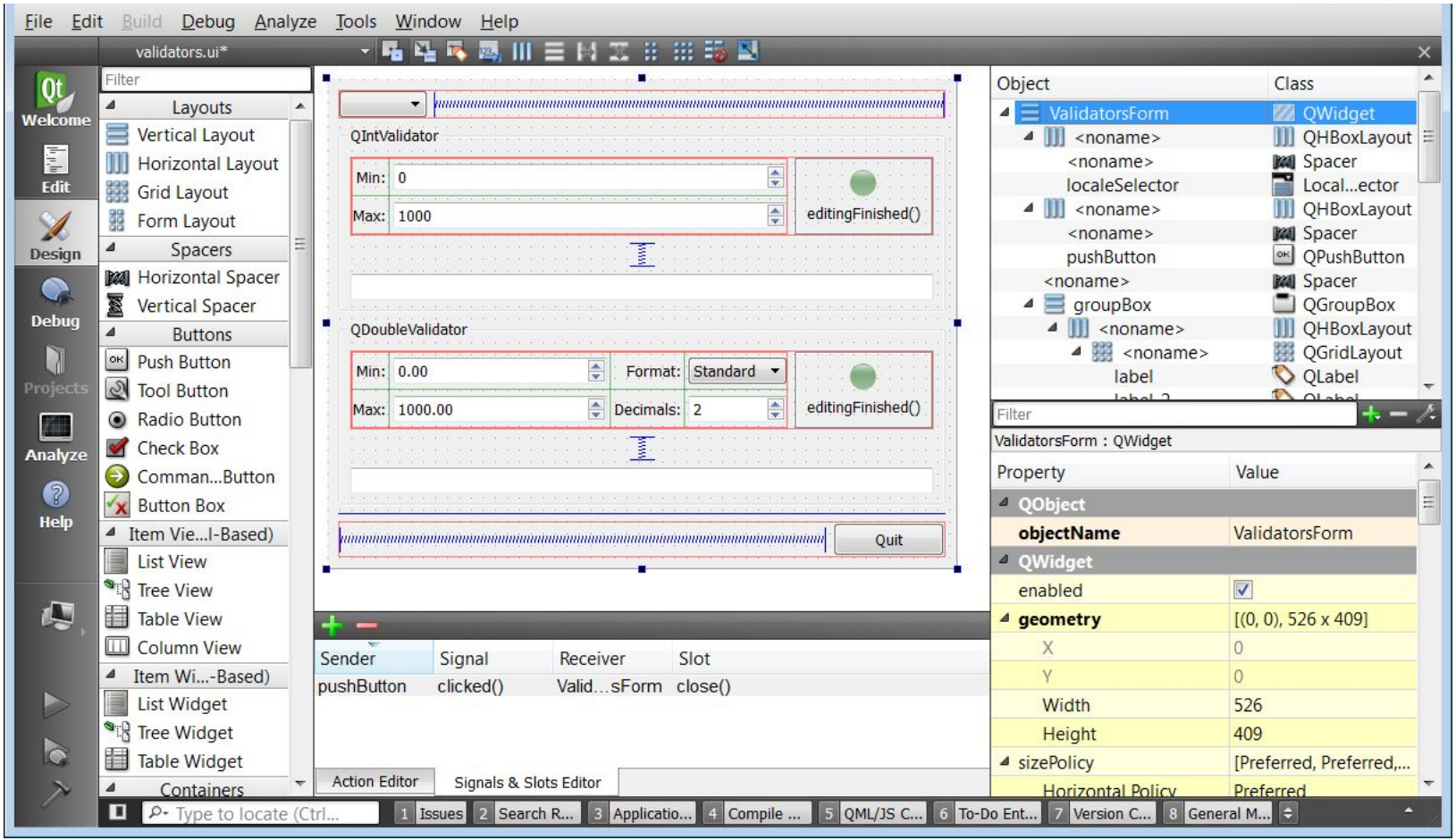
- Los IDE colaboran con el desarrollo de aplicaciones integrando varias herramientas en un solo programa.
- En general, existen IDEs específicas para cada lenguaje de programación.
- Algunas IDEs se pueden adaptar para programar en varios lenguajes, como Eclipse, Netbeans o Jetbrains (que ofrece diferentes versiones de casi un mismo software para diversos lenguajes).


File Edit Source Refactor Navigate Search Run Project Window Help
C/C++
Team Syn...
Project Explorer
HelloWorld
Binaries
src
HelloWorld.cpp
Debug
HelloWorld.cpp
1 //==
2 // Name : HelloWorld.cpp
3 // Author :
4 // Version :
5 // Copyright : Your copyright notice
6 // Description : Hello World in C++, Ansi-style
7 //==
8
9 #include <iostream>
10 using namespace std;
11
12 int main() {
13 cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
14 return 0;
15 }
Outlin
Make
iostream
std
main() : int
Problems
Tasks
Console
Properties
Valgrind
Search
CDT Build Console [HelloWorld]
**** Build of configuration Debug for project HelloWorld ****
make all
Building file: ../src/HelloWorld.cpp
Invoking: GCC C++ Compiler
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/HelloWorld.d" -MT"src/HelloWorld.d" -o "src/HelloWorld.o" "../src/HelloWorld.cpp"
Finished building: ../src/HelloWorld.cpp
Building target: HelloWorld
Invoking: GCC C++ Linker
g++ -o "HelloWorld" ./src/HelloWorld.o
Finished building target: HelloWorld
**** Build Finished ****

IDE Eclipse


Quote_1 - NetBeans IDE Dev 20110419-e74a70644161
File Edit View Navigate Source Refactor Run Debug Team Tools Window Help
Debug
Projects
Classes
Services
Quote_1
unnamed
Cpu
Customer
Disk
Memory
Module
Stack<T>
Stack<double>
Stack<int>
System
main(int argc, char** argv)
operator <<(ostream& , const Customer&)
operator <<(ostream& , const Module&)
operator <<(ostream& , const System&)
C:\cygwin\lib\gcc\i686-pc-cygwin\3.4.4\include
C:\cygwin\lib\gcc\i686-pc-cygwin\3.4.4\include\c++
C:\cygwin\usr\include
Stack<T> - Navigator
NUM_PARAMS
Stack<T>
Stack<double>
Stack<int>
cpu.h
cstdlib
customer.h
disk.h
iostream
list
main(int argc, char** argv)
memory.h
system.h
Start Page
memory.h
quote.cc
module.h
} // End of local namespace
template<T> class Stack {
Has specialization ...
Stack<double>
Stack<int>
public:
T pop();
};
template<> class Stack<int> {
public:
int pop();
};
template<> class Stack<double> {
public:
int pop();
};
int main(int argc, char** argv) {
if (argc != NUM_PARAMS) {
return EXIT_FAILURE;
}
cout << "Support metric quote program" << endl << endl;
fetchCustomersList();
int discount = -1;
quote.cc - Hierarchy
quote.cc
cpu.h
cstdlib
customer.h
iostream
c++config.h
istream
ostream
disk.h
iostream
list
memory.h
system.h
Macro Expansion of quote.cc
int main(int argc, char** argv) {
if (argc != 4) {
return 1;
}
cout << "Support metric quote program" << endl << endl;
fetchCustomersList();
int discount = -1;
string customerName;
do {
outCustomersList();
cout << "Enter customer name: ";
Number of expansions: 3
Terminal - vv159170@192.168.1.33
vv159170@Lenovo:~/devarea$ la
total 108
drwxr-xr-x 8 vv159170 uucp 4096 2011-03-27 22:49 .
drwxr-xr-x 74 vv159170 uucp 4096 2011-04-19 22:29 ..
-rw-r--r-- 1 vv159170 uucp 0 2011-03-13 23:17 i18n-check.log
drwxr-xr-x 12 vv159170 uucp 4096 2010-12-20 10:51 java
-rw-r--r-- 1 vv159170 uucp 315 2010-09-05 22:10 netbeans.clusters
-rw-r--r-- 1 vv159170 uucp 295 2011-04-19 21:40 netbeans.clusters.orig
lrwxrwxrwx 1 vv159170 uucp 5 2010-12-03 23:59 opt -> /opt/
drwxr-xr-x 14 vv159170 uucp 4096 2011-04-07 17:20 osprojects
drwxr-xr-x 896 vv159170 uucp 36864 2011-04-07 17:15 release70
drwxr-xr-x 23 vv159170 uucp 4096 2011-04-19 22:01 sputnik
drwxr-xr-x 5 vv159170 uucp 4096 2010-12-04 00:57 toolshg
drwxr-xr-x 785 vv159170 uucp 36864 2011-04-19 20:11 trunk
vv159170@Lenovo:~/devarea$
113 | 1
INS

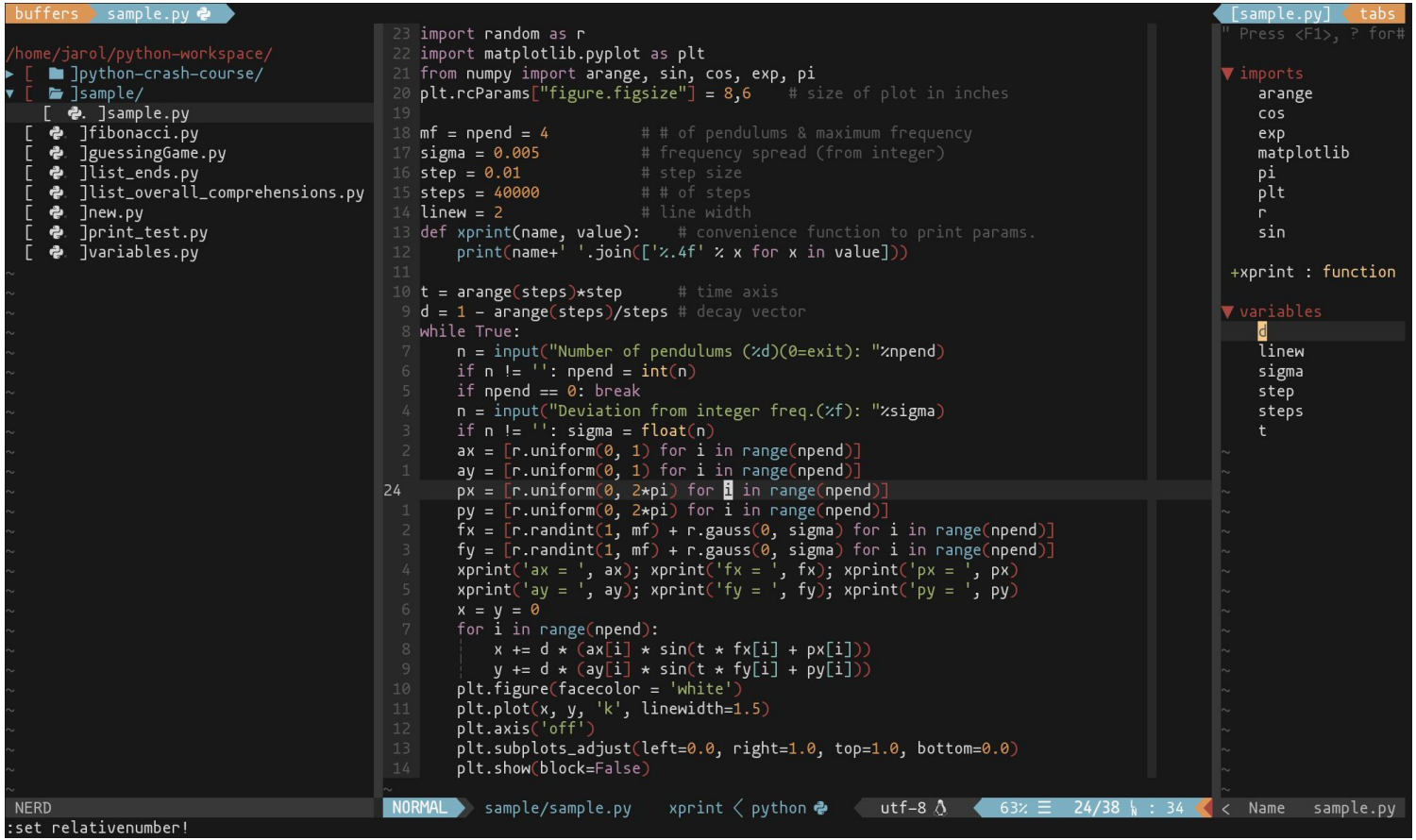
Netbeans


File Edit Build Debug Analyze Tools Window Help
validators.ui*
Welcome
Edit
Design
Debug
Projects
Analyze
Help
Filter
Layouts
Vertical Layout
Horizontal Layout
Grid Layout
Form Layout
Spacers
Horizontal Spacer
Vertical Spacer
Buttons
Push Button
Tool Button
Radio Button
Check Box
Comman...Button
Button Box
Item Vie...l-Based)
List View
Tree View
Table View
Column View
Item Wi...-Based)
List Widget
Tree Widget
Table Widget
Containers
QIntValidator
Min: 0
Max: 1000
editingFinished()
QDoubleValidator
Min: 0.00
Max: 1000.00
Format: Standard
Decimals: 2
editingFinished()
Quit
Sender Signal Receiver Slot
pushButton clicked() Valid...sForm close()
Action Editor
Signals & Slots Editor
Object Class
ValidatorsForm QWidget
<noname> QHBoxLayout
<noname> Spacer
localeSelector Local...ector
<noname> QHBoxLayout
<noname> Spacer
pushButton QPushButton
<noname> Spacer
groupBox QGroupBox
<noname> QHBoxLayout
<noname> QGridLayout
label QLabel
Filter
ValidatorsForm : QWidget
Property Value
QObject
objectName ValidatorsForm
QWidget
enabled
geometry [(0, 0), 526 x 409]
X 0
Y 0
Width 526
Height 409
sizePolicy [Preferred, Preferred,...
Horizontal Policy Preferred
Type to locate (Ctrl...
1 Issues
2 Search R...
3 Applicatio...
4 Compile ...
5 QML/JS C...
6 To-Do Ent...
7 Version C...
8 General M...

QT Creator

Desarrollo sin una IDE

- También es posible desarrollar software sin utilizar una IDE, conectando los distintos componentes entre sí a mano.
- Se pueden combinar distintos programas como:
  - **Editor de textos:** Vim, Emacs, Notepad++, SublimeText, etc.
  - **Compiladores:** GCC, LLVM Clang o ICC
  - **Debugger:** GDB
  - **Gestor de dependencias:** Make o CMake



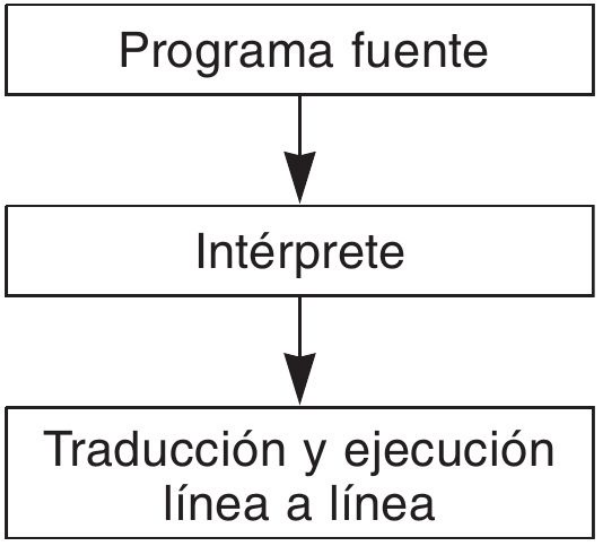
Vim configurado para desarrollar con Python

Traductores de lenguaje

- El proceso de traducción de un programa fuente escrito en un lenguaje de alto nivel a un lenguaje máquina comprensible por la computadora, se realiza mediante programas llamados “traductores”.
- Los traductores de lenguaje son programas que traducen a su vez los programas fuente escritos en lenguajes de alto nivel a código máquina.
- Los traductores se dividen en compiladores e intérpretes.

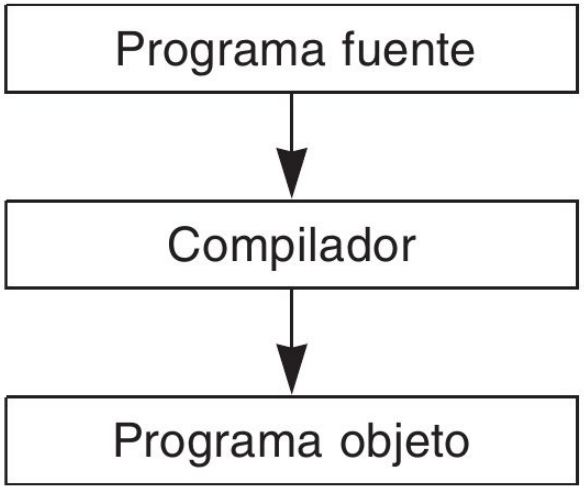
Interpretes

- Un intérprete es un traductor que toma un programa fuente, lo traduce y, a continuación, lo ejecuta.
- El sistema de traducción consiste en:
  1. traducir la primera sentencia del programa a lenguaje máquina,
  2. se detiene la traducción,
  3. se ejecuta la sentencia;
  4. a continuación, se traduce la siguiente sentencia,
  5. se detiene la traducción,
  6. se ejecuta la sentencia
  7. y así sucesivamente hasta terminar el programa
- Lenguajes de programación interpretados: Python, Ruby, PHP, Perl, entre otros.



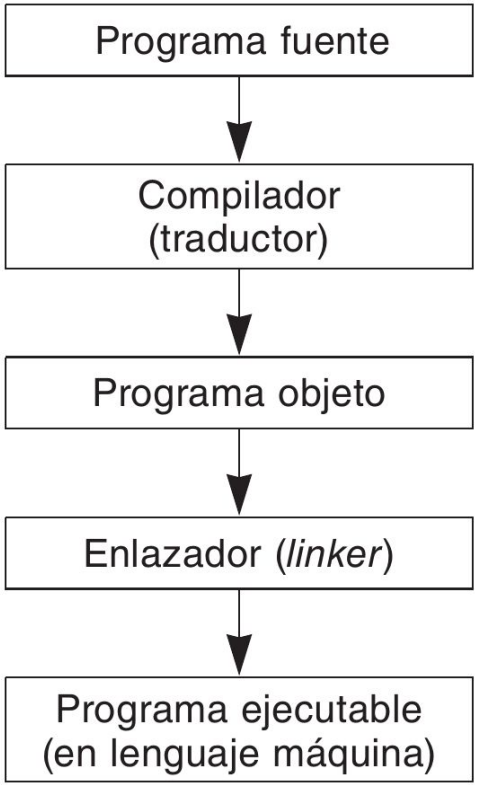
Compiladores

- Un compilador es un programa que traduce los programas fuente escritos en lenguaje de alto nivel a lenguaje máquina.
- La traducción del programa completo se realiza en una sola operación denominada compilación del programa; es decir, se traducen todas las instrucciones del programa en un solo bloque.
- El programa compilado y depurado (eliminados los errores del código fuente) se denomina programa ejecutable porque ya se puede ejecutar directamente y cuantas veces se desee
- sólo deberá volver a compilarse de nuevo en el caso de que se modifique alguna instrucción del programa.
- De este modo el programa ejecutable no necesita del compilador para su ejecución. Los traductores de lenguajes típicos más utilizados son: C, C++, C#, Pascal, Go, Rust.



La compilación y sus fases

- La compilación es el proceso de traducción de programas fuente a programas objeto
- El programa objeto obtenido de la compilación ha sido traducido normalmente a código máquina
- Para conseguir el programa máquina real se debe utilizar un programa llamado montador o enlazador (linker)
- El proceso de montaje conduce a un programa en lenguaje máquina directamente ejecutable

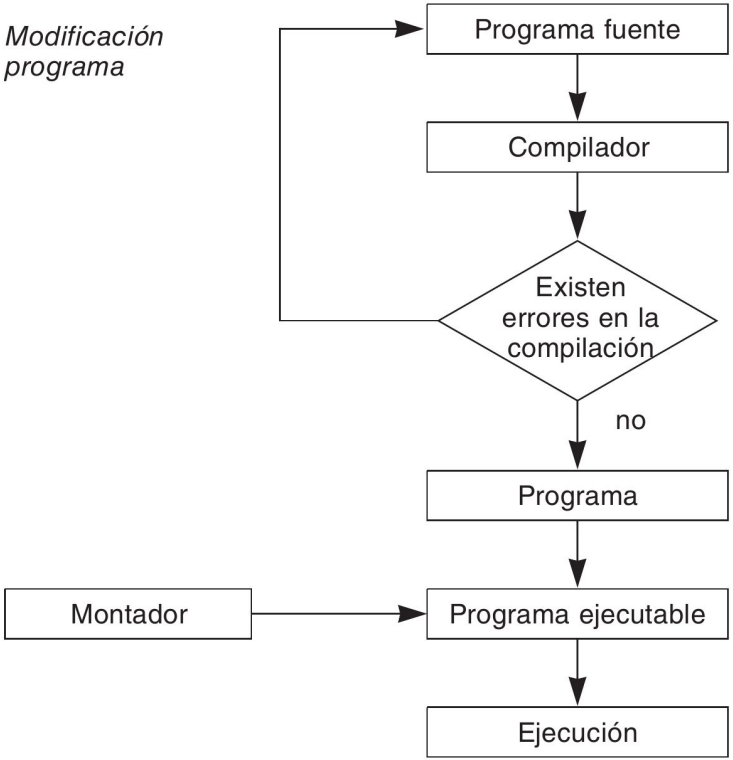


La compilación y sus fases

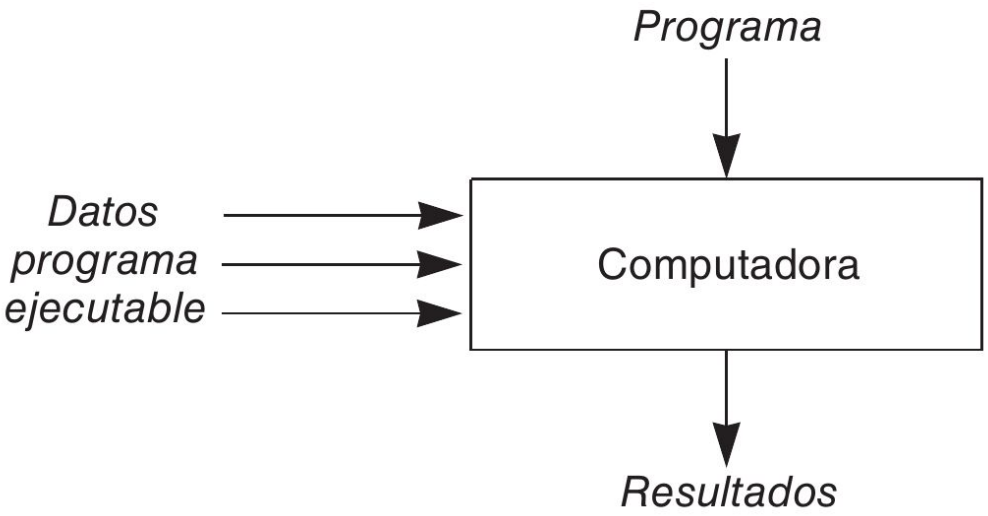
El proceso de ejecución de un programa escrito en un lenguaje de programación y mediante un compilador suele tener los siguientes pasos:

1. Escritura del programa fuente con un editor de texto (o IDE) y guardarlo en un dispositivo de almacenamiento (por ejemplo, un disco)
2. Introducir el programa fuente en memoria
3. Compilar el programa con el compilador seleccionado
4. Verificar y corregir errores de compilación (listado de errores)
5. Obtención del programa objeto
6. El enlazador (linker) obtiene el programa ejecutable
7. Se ejecuta el programa y, si no existen errores, se tendrá la salida del programa

Fases de ejecución de un programa



Ejecución de un programa



Fases en la resolución de un problema

Fases en la resolución de problemas

- El proceso de resolución de un problema con una computadora conduce a la escritura de un programa y a su ejecución en la misma.
- Aunque el proceso de diseñar programas es, esencialmente, un proceso creativo, se puede considerar una serie de fases o pasos comunes, que generalmente deben seguir todos los programadores.
- Las fases de resolución de un problema con computadora son:
    - Análisis del problema
    - Diseño del algoritmo
    - Codificación
    - Compilación y ejecución
    - Verificación
    - Depuración
    - Mantenimiento
    - Documentación

Resolución de problemas

- **Análisis**: el problema se analiza teniendo presente la especificación de los requisitos dados por el cliente de la empresa o por la persona que encarga el programa.
- **Diseño**: Una vez analizado el problema, se diseña una solución que conducirá a un algoritmo que resuelva el problema.
- **Codificación** (implementación): La solución se escribe en la sintaxis del lenguaje de alto nivel (por ejemplo, C) y se obtiene un programa fuente que se compila a continuación.

Resolución de problemas

- **Ejecución, verificación y depuración:** El programa se ejecuta, se comprueba rigurosamente y se eliminan todos los errores (denominados “bugs”, en inglés) que puedan aparecer.
- **Mantenimiento:** El programa se actualiza y modifica, cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de sus usuarios.
- **Documentación:** Escritura de las diferentes fases del ciclo de vida del software, esencialmente el análisis, diseño y codificación, unidos a manuales de usuario y de referencia, así como normas para el mantenimiento

Resolución de problemas

- Las dos primeras fases conducen a un diseño detallado escrito en forma de algoritmo.
- Durante la tercera fase (codificación) se implementa el algoritmo en un código escrito en un lenguaje de programación, reflejando las ideas desarrolladas en las fases de análisis y diseño.
- Las fases de compilación y ejecución traducen y ejecutan el programa.
- En las fases de verificación y depuración el programador busca errores de las etapas anteriores y los elimina.
- Comprobará que mientras más tiempo se gaste en la fase de análisis y diseño, menos se gastará en la depuración del programa.
- Por último, se debe realizar la documentación del programa.

¿Qué es un Algoritmo?

Un algoritmo es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos:

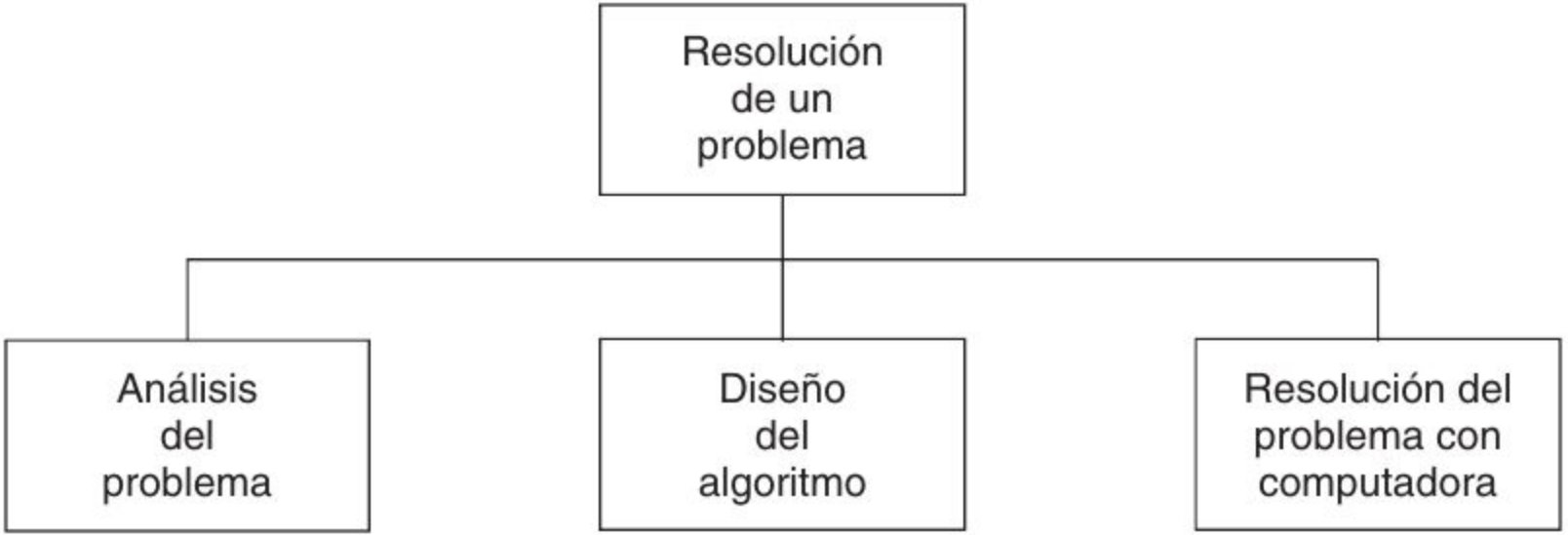
- **Preciso**: indica el orden de realización en cada paso
- **Definido**: si se sigue dos veces, obtiene el mismo resultado cada vez
- **Finito**: tiene fin; un número determinado de pasos

Algoritmos vs Heurística

- Los métodos que utilizan algoritmos se denominan métodos algorítmicos, en oposición a los métodos que implican algún juicio o interpretación que se denominan métodos heurísticos.
- Los métodos algorítmicos se pueden implementar en computadoras; sin embargo, los procesos heurísticos no han sido convertidos fácilmente en las computadoras
- Las técnicas de inteligencia artificial han hecho posible la implementación del proceso heurístico en computadoras.
- Ejemplos de algoritmos son: instrucciones para montar en una bicicleta, hacer una receta de cocina, obtener el máximo común divisor de dos números, etc.
- Los algoritmos se pueden expresar por fórmulas, diagramas de flujo y pseudocódigos.
- Esta última representación es la más utilizada para su uso con lenguajes estructurados como C

Problema para resolver

Pasos a seguir



Analizar un problema

Para poder identificar y definir bien un problema es conveniente responder a las siguientes preguntas:

- ¿Qué entradas se requieren?: tipo de datos con los cuales se trabaja y cantidad
- ¿Cuál es la salida deseada?: tipo de datos de los resultados y cantidad
- ¿Qué método produce la salida deseada?
- Requisitos o requerimientos adicionales y restricciones a la solución.

Problema

Se desea obtener una tabla con las depreciaciones acumuladas y los valores reales de cada año, de un automóvil comprado por 20.000 euros en el año 2005, durante los seis años siguientes suponiendo un valor de recuperación de 2.000. Realizar el análisis del problema, conociendo la fórmula de la depreciación anual constante D para cada año de vida útil.

$$D = \frac{\text{coste} - \text{valor de recuperación}}{\text{vida útil}}$$

$$D = \frac{20.000 - 2.000}{6} = \frac{18.000}{6} = 3.000$$

El valor de recuperación es la fracción del costo inicial que se elimina por depreciación cada año

Entradas, proceso y salidas

Entrada	coste original vida útil valor de recuperación
Salida	depreciación anual por año depreciación acumulada en cada año valor del automóvil en cada año
Proceso	depreciación acumulada cálculo de la depreciación acumulada cada año cálculo del valor del automóvil en cada año

Salidas esperadas

Año	Depreciación	Depreciación acumulada	Valor anual
1 (2006)	3.000	3.000	17.000
2 (2007)	3.000	6.000	14.000
3 (2008)	3.000	9.000	11.000
4 (2009)	3.000	12.000	8.000
5 (2010)	3.000	15.000	5.000
6 (2011)	3.000	18.000	2.000

Diseño del algoritmo

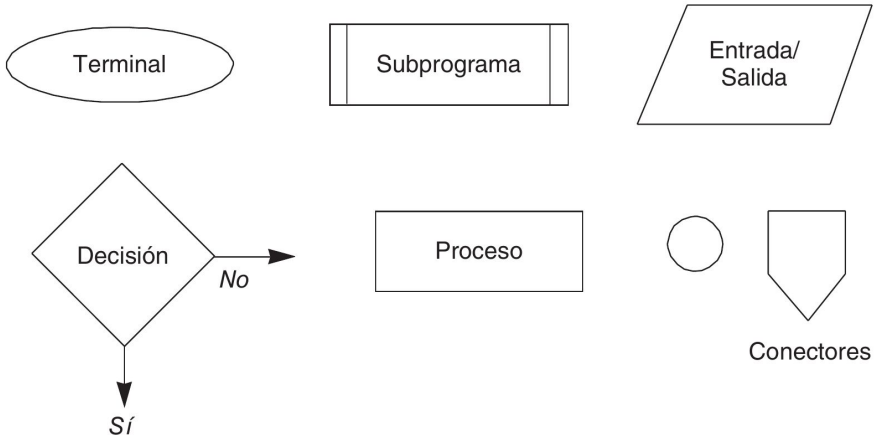
- En la etapa de análisis del proceso de programación se determina qué hace el programa.
- En la etapa de diseño se determina cómo hace el programa la tarea solicitada.
- Los métodos más eficaces para el proceso de diseño se basan en el conocido **“divide y vencerás”**
- Es decir, la resolución de un problema complejo se realiza dividiendo el problema en subproblemas y a continuación dividiendo estos subproblemas en otros de nivel más bajo, hasta que pueda ser implementada una solución en la computadora.
- Este método se conoce técnicamente como diseño descendente (top-down) o modular.
- El proceso de romper el problema en cada etapa y expresar cada paso en forma más detallada se denomina refinamiento sucesivo.
- Cada subprograma es resuelto mediante un módulo (subprograma) que tiene un solo punto de entrada y un solo punto de salida.

Diseño del algoritmo

- Cualquier programa bien diseñado consta de un programa principal (el módulo de nivel más alto) que llama a subprogramas (módulos de nivel más bajo) que a su vez pueden llamar a otros subprogramas.
- Este método de diseñar un programa se lo denomina “programación modular”
- Los módulos pueden ser planeados, codificados, comprobados y depurados independientemente (incluso por diferentes programadores) y a continuación combinarlos entre sí.
- El proceso implica la ejecución de los siguientes pasos hasta que el programa se termina:
    1. Programar un módulo.
    2. Comprobar el módulo.
    3. Si es necesario, depurar el módulo.
    4. Combinar el módulo con los módulos anteriores.

Herramientas de programación y diseño

- Las dos herramientas más utilizadas comúnmente para diseñar algoritmos son: diagramas de flujo y pseudocódigos.
- Un diagrama de flujo (flowchart) es una representación gráfica de un algoritmo. Los símbolos utilizados han sido normalizados por el Instituto Norteamericano de Normalización (ANSI)

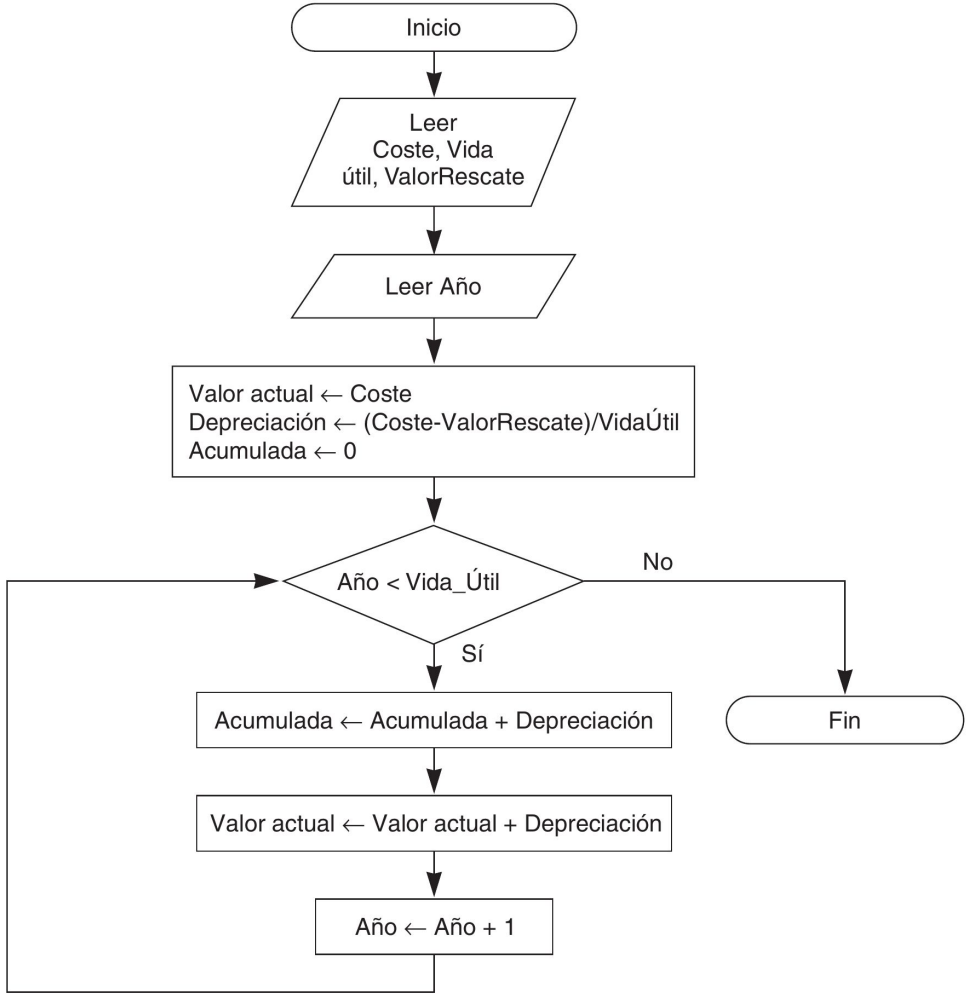


pseudocodigo

- El pseudocódigo es una herramienta de programación en la que las instrucciones se escriben en palabras similares al inglés o español, que facilitan tanto la escritura como la lectura de programas.
- En esencia, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.
- No existen reglas para escritura del pseudocódigo en español, según la bibliografía que se consulte podrá encontrar variaciones en el pseudocódigo utilizado.

```
Previsiones de depreciacion
Introducir coste
           vida util
           valor final de rescate (recuperacion)
imprimir cabeceras
Establecer el valor inicial del año
Calcular depreciacion
mientras valor año =< vida util hacer
      calcular depreciacion acumulada
      calcular valor actual
      imprimir una linea en la tabla
      incrementar el valor del año
fin de mientras
```

Diagrama de flujo o Flowchart



Otro ejemplo de pseudocódigo

- Calcular el valor de la suma 1+2+3+...+100.

  Se utiliza una variable Contador como un contador que genere los sucesivos números enteros, y Suma para almacenar las sumas parciales 1, 1+2, 1+2+3...

```
1. Establecer Contador a 1
2. Establecer Suma a 0
3. mientras Contador <= 100 hacer
       Sumar Contador a Suma
       Incrementar Contador en 1
   fin_mientras
4. Visualizar Suma
```

Codificación de un programa

- La codificación es la escritura en un lenguaje de programación de la representación del algoritmo desarrollada en las etapas precedentes.
- Dado que el diseño de un algoritmo es independiente del lenguaje de programación utilizado para su implementación, el código puede ser escrito con igual facilidad en un lenguaje o en otro.
- Para realizar la conversión del algoritmo en programa se deben sustituir las palabras reservadas en español por sus homónimos en inglés, y las operaciones/instrucciones indicadas en lenguaje natural por el lenguaje de programación correspondiente.

```
{Este programa obtiene una tabla de depreciaciones
 acumuladas y valores reales de cada año de un
 determinado producto}

algoritmo primero
  Real: Coste, Depreciacion,
        Valor_Recuperacion
        Valor_Actual,
        Acumulado
        Valor_Anual;
  entero: Año, Vida_Util;
  inicio
        escribir('introduzca coste, valor recuperación y vida útil')
        leer(Coste, Valor_Recuperacion, Vida_Util)
        escribir('Introduzca año actual')
        leer(Año)
        Valor_Actual ← Coste;
        Depreciacion ← (Coste-Valor_Recuperacion)/Vida_Util
        Acumulado ← 0
        escribir('Año Depreciación Dep. Acumulada')
        mientras (Año < Vida_Util)
           Acumulado ← Acumulado + Depreciacion
           Valor_Actual ← Valor_Actual - Depreciacion
           escribir('Año, Depreciacion, Acumulado')
           Año ← Año + 1;
        fin mientras
  fin
```

Documentación interna

- La documentación de un programa se clasifica en interna y externa.
- La documentación interna es la que se incluye dentro del código del programa fuente mediante comentarios que ayudan a la comprensión del código.
- Todas las líneas de programas que comiencen con un símbolo /* o // son comentarios (en C).
- El programa no los necesita y la computadora los ignora.
- Estas líneas de comentarios sólo sirven para hacer los programas más fáciles de comprender.
- El objetivo del programador debe ser escribir códigos sencillos y limpios.
- Debido a que las máquinas actuales soportan grandes memorias (8/16/32 GB de memoria RAM en PC) no es necesario recurrir a técnicas de ahorro de memoria, por lo que es recomendable que se incluya el mayor número de comentarios posibles, pero eso sí, que sean significativos.

Compilación y ejecución de un programa

- Una vez que el algoritmo se ha convertido en un programa fuente, es necesario ingresarlo en un editor de textos o IDE.
- El programa fuente debe ser traducido a lenguaje máquina, este proceso se realiza con el compilador y el sistema operativo que se encarga prácticamente de la compilación
- Si tras la compilación se presentan errores (errores de compilación) en el programa fuente, es preciso volver a editar el programa, corregir los errores y compilar de nuevo.
- Este proceso se repite hasta que no se producen errores, obteniéndose el programa objeto que todavía no es ejecutable directamente.

Compilación y ejecución de un programa

- Suponiendo que no existen errores en el programa fuente, se debe instruir al sistema operativo para que realice la fase de montaje o enlace (link), carga, del programa objeto con las bibliotecas del programa del compilador.
- El proceso de montaje produce un programa ejecutable.
- Una vez que el programa ejecutable se ha creado, ya se puede ejecutar (correr) desde el sistema operativo con sólo teclear su nombre.
- Suponiendo que no existen errores durante la ejecución (llamados errores en tiempo de ejecución), se obtendrá la salida de resultados del programa.
- Las instrucciones u órdenes para compilar y ejecutar un programa en C, C++,... o cualquier otro lenguaje dependerá de su entorno de programación y del sistema operativo en que se ejecute Windows, Linux, Unix, etc.

Verificación y depuración de un programa

- La **verificación** o compilación de un programa es el proceso de ejecución del programa con una amplia variedad de datos de entrada, llamados datos de test o prueba, que determinarán si el programa tiene o no errores (“bugs”)
- Para realizar la **verificación** se debe desarrollar una amplia gama de datos de test: valores normales de entrada, valores extremos de entrada que comprueben los límites del programa y valores de entrada que comprueben aspectos especiales del programa
- La **depuración** es el proceso de encontrar los errores del programa y corregir o eliminar dichos errores
- Cuando se ejecuta un programa, se pueden producir tres tipos de errores: errores de compilación, errores de ejecución y errores lógicos.

Verificación y depuración de un programa

1. **Errores de compilación**: se producen normalmente por un uso incorrecto de las reglas del lenguaje de programación y suelen ser errores de sintaxis. Si existe un error de sintaxis, la computadora no puede comprender la instrucción, no se obtendrá el programa objeto y el compilador imprimirá una lista de todos los errores encontrados durante la compilación.

2. **Errores de ejecución**: estos errores se producen por instrucciones que la computadora puede comprender pero no ejecutar. Ejemplos típicos son: división por cero y raíces cuadradas de números negativos. En estos casos se detiene la ejecución del programa y se imprime un mensaje de error.

3. **Errores lógicos**: se producen en la lógica del programa y la fuente del error suele ser el diseño del algoritmo. Estos errores son los más difíciles de detectar, ya que el programa puede funcionar y no producir errores de compilación ni de ejecución, y sólo puede advertirse el error por la obtención de resultados incorrectos. En este caso se debe volver a la fase de diseño del algoritmo, modificar el algoritmo, cambiar el programa fuente y compilar y ejecutar una vez más.

Documentación y mantenimiento

- La documentación de un problema consta de las descripciones de los pasos a dar en el proceso de resolución de dicho problema.
- La importancia de la documentación debe ser destacada por su decisiva influencia en el producto final.
- Programas pobremente documentados son difíciles de leer, más difíciles de depurar y casi imposibles de mantener y modificar.
- La documentación de un programa puede ser interna y externa. La documentación interna es la contenida en líneas de comentarios.
- La documentación externa incluye análisis, diagramas de flujo y/o pseudocódigos, manuales de usuario con instrucciones para ejecutar el programa y para interpretar los resultados.
- La documentación es vital cuando se desea corregir posibles errores futuros o bien cambiar el programa. Tales cambios se denominan mantenimiento del programa. Después de cada cambio la documentación debe ser actualizada para facilitar cambios posteriores.

Modelos de programación

Programación modular

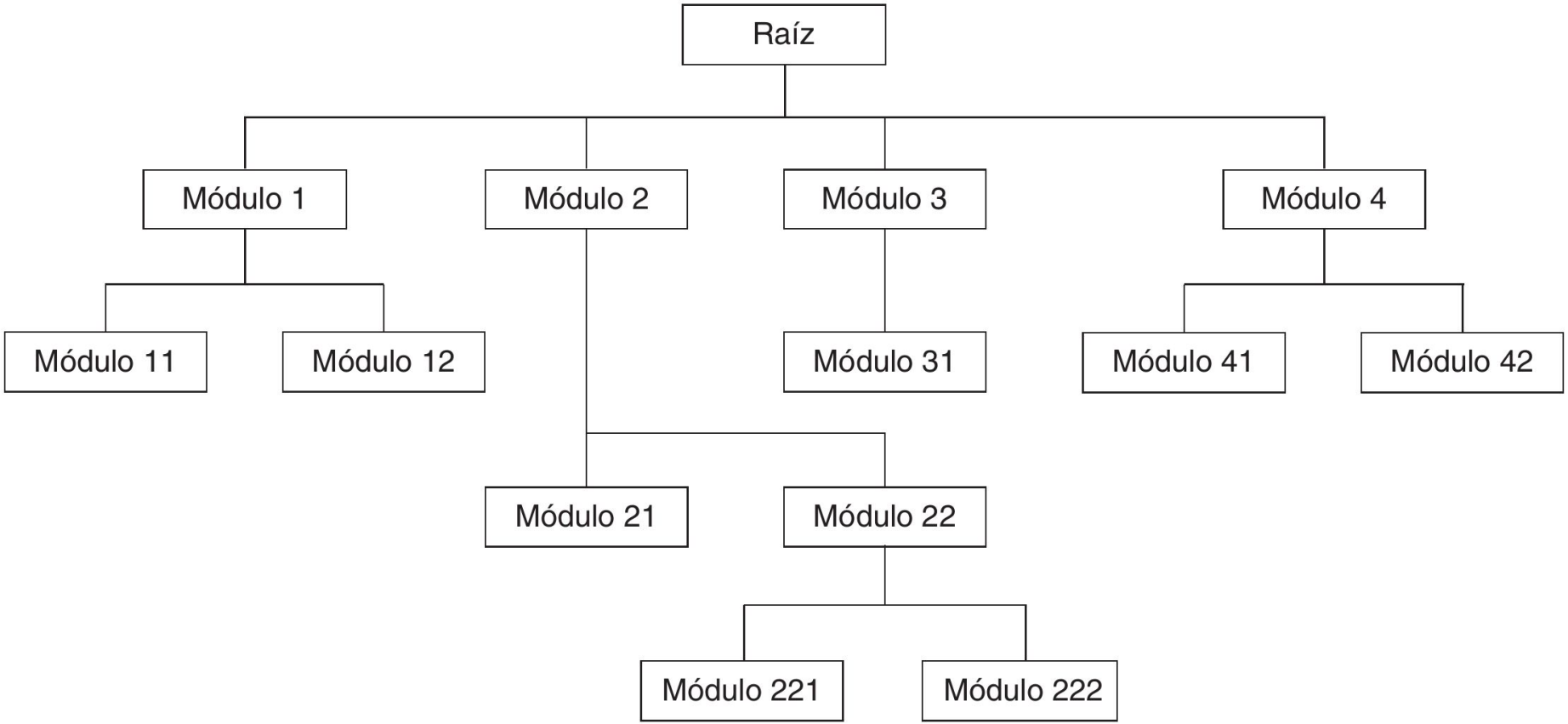
- En programación modular el programa se divide en módulos (partes independientes), cada uno de los cuales ejecuta una única actividad o tarea y se codifican independientemente de otros módulos.
- Cada uno de estos módulos se analiza, codifica y pone a punto por separado.
- Cada programa contiene un módulo denominado programa principal que controla todo lo que sucede; se transfiere el control a submódulos o subprogramas, de modo que ellos puedan ejecutar sus funciones; sin embargo, cada submódulo devuelve el control al módulo principal cuando se haya completado su tarea.
- Si la tarea asignada a cada submódulo es demasiado compleja, éste deberá romperse en otros módulos más pequeños.
- El proceso sucesivo de subdivisión de módulos continúa hasta que cada módulo tenga solamente una tarea específica que ejecutar

Programación modular

- Esta tarea puede ser entrada, salida, manipulación de datos, control de otros módulos o alguna combinación de éstos.
- Un módulo puede transferir temporalmente (bifurcar) el control a otro módulo; sin embargo, cada módulo debe eventualmente devolver el control al módulo del cual se recibe originalmente el control.
- Los módulos son independientes en el sentido en que ningún módulo puede tener acceso directo a cualquier otro módulo excepto el módulo al que llama y sus propios submódulos.
- Los resultados producidos por un módulo pueden ser utilizados por cualquier otro módulo cuando se transfiera a ellos el control

Programación modular

- Dado que los módulos son independientes, diferentes programadores pueden trabajar simultáneamente en diferentes partes del mismo programa.
- Esto reducirá el tiempo del diseño del algoritmo y posterior codificación del programa.
- Un módulo se puede modificar radicalmente sin afectar a otros módulos, incluso sin alterar su función principal.
- La descomposición de un programa en módulos independientes más simples se conoce también como el método de **divide y vencerás** (divide and conquer).
- Cada módulo se diseña con independencia de los demás, y siguiendo un método ascendente o descendente se llegará hasta la descomposición final del problema en módulos en forma jerárquica.


Raíz
Módulo 1
Módulo 2
Módulo 3
Módulo 4
Módulo 11
Módulo 12
Módulo 31
Módulo 41
Módulo 42
Módulo 21
Módulo 22
Módulo 221
Módulo 222

Programación Estructurada

- C, Pascal, FORTRAN, y lenguajes similares, se conocen como lenguajes procedimentales (por procedimientos).
- Cada sentencia o instrucción señala al compilador para que realice alguna tarea: obtener una entrada, producir una salida, sumar tres números, dividir por cinco, etc. En resumen, un programa en un lenguaje procedimental es un conjunto de instrucciones o sentencias.
- En el caso de pequeños programas, estos principios de organización (denominados paradigma) se demuestran eficientes.
- El programador sólo tiene que crear esta lista de instrucciones en un lenguaje de programación, compilar en la computadora y ésta, a su vez, ejecuta estas instrucciones.

Programación Estructurada

- Cuando los programas se vuelven más grandes, la lista de instrucciones aumenta considerablemente, de modo tal que el programador tiene muchas dificultades para controlar ese gran número de instrucciones.
- Los programadores pueden controlar, de modo normal, unos centenares de líneas de instrucciones. Para resolver este problema los programas se descompusieron en unidades más pequeñas que adoptaron el nombre de funciones (procedimientos, subprogramas o subrutinas en otros lenguajes de programación).
- De este modo en un programa orientado a procedimientos se divide en funciones, de modo que cada función tiene un propósito bien definido y resuelve una tarea concreta, y se diseña una interfaz claramente definida (el prototipo o cabecera de la función) para su comunicación con otras funciones.

Programación Estructurada

- Con el paso de los años, la idea de romper el programa en funciones fue evolucionando y se llegó al agrupamiento de las funciones en otras unidades más grandes llamadas módulos (normalmente, en el caso de C, denominadas archivos o ficheros)
- El principio seguía siendo el mismo: agrupar componentes que ejecutan listas de instrucciones (sentencias).
- Esta característica hace que a medida que los programas se hacen más grandes y complejos, el paradigma estructurado comienza a dar señales de debilidad y resultando muy difícil terminar los programas de un modo eficiente

Programación orientada a objetos

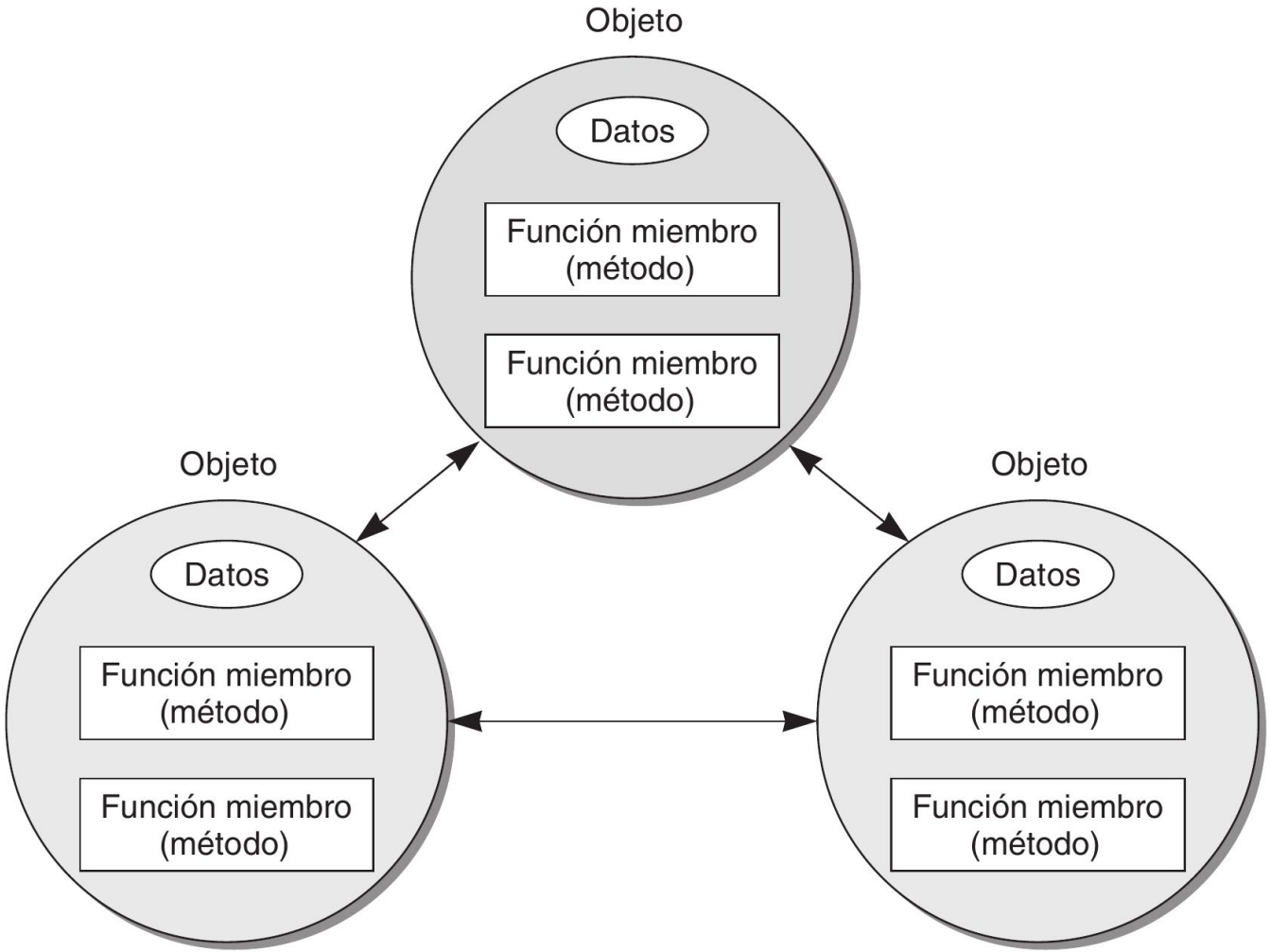
- La programación orientada a objetos, es tal vez el paradigma de programación más utilizado en el mundo del desarrollo de software y de la ingeniería de software del siglo XXI.
- Al contrario que la programación procedimental que enfatiza en los algoritmos, la POO se enfoca en los datos.
- En lugar de intentar ajustar un problema al enfoque procedimental de un lenguaje, POO intenta ajustar el lenguaje al problema.
- La idea es diseñar formatos de datos que se correspondan con las características esenciales de un problema.
- La idea fundamental de los lenguajes orientados a objetos es combinar en una única unidad o módulo, tanto los **datos** como las **funciones** que operan sobre esos datos, a esta unidad se la llama un **objeto**.

Programación orientada a objetos

- Las funciones de un objeto se llaman funciones miembro en C++ o métodos (éste es el caso de Smalltalk, uno de los primeros lenguajes orientados a objetos), y son el único medio para acceder a sus datos.
- Los datos de un objeto, se conocen también como atributos o variables de instancia.
    - Si se desea leer datos de un objeto, se llama a una función miembro del objeto.
    - Se accede a los datos y se devuelve un valor.
    - No se puede acceder a los datos directamente.
    - Los datos están ocultos, de modo que están protegidos de alteraciones accidentales.
    - Los datos y las funciones se dice que están encapsulados en una única entidad.

Programación orientada a objetos

- El encapsulamiento de datos y la ocultación de los datos son términos clave en la descripción de lenguajes orientados a objetos.
- Si se desea modificar los datos de un objeto, se conoce exactamente cuáles son las funciones que interactúan con miembros del objeto.
- Ninguna otra función puede acceder a los datos.
- Esto simplifica la escritura, depuración y mantenimiento del programa.
- Un programa C++ se compone normalmente de un número de objetos que se comunican unos con otros mediante la llamada a otras funciones miembro.
- La llamada a una función miembro de un objeto se denomina enviar un mensaje a otro objeto


Objeto
Datos
Función miembro (método)
Función miembro (método)
Objeto
Datos
Función miembro (método)
Función miembro (método)
Objeto
Datos
Función miembro (método)
Función miembro (método)

Referencias:

- Fundamentos de programación: algoritmos, estructuras de datos y objetos. 4ta Edicion. Luis Joyanes Aguilar. McGraw Hill. 2008
- Joyanes Aguilar, Luis. Fundamentos de la programación. McGraw-Hill /Interamericana de España. 2013
- Deitel Harvey y Deitel Paul. Como programar en C/C++. 9na Edición. Addison-Wesley. 2014